Завдання Для початку реалізувати енкодер та декодер (плеєр) для відеоряду на основі методів оцінки руху. Відповідно вхідним файлом буде відеозапис (наприклад трейлер до Тенет), одним з виходів буде файлик у вашому форматі що містить гіпотетично стиснений відеоряд з наприклад інформації щодо руху кожного третього кадру і два сусідні у повному розмірі (MPEG-7 алгоритми з що ми їх дотично згадували в попередніх лекціях будуть в нагоді). Другим виходом буде власне відтворення відео з вашого файлу. Після того як ви налаштували ваш енкодер і плеєр, йдете до друга по команді що в цей час налаштовував свої енкодер і плеєр на основі іншого алгоритму апроксимації руху і ви починаєте об'єднувати зусилля: користуючись вашими енкодерами ви генеруєте ваші апроксимовані зображення та обчислюєте їх різницю в порівнянні з дійсним зображеннями з оригінального відеоряду, збираючи при цьому візуальні різниці для кожного з методів. Окрім цього слід зібрати метрику швидкодії енкодерів і плеєрів, порівняти отримані результати і залляти їх аналогічним чином в гуглдок. Бонусні бали за опцію datamoshing і запис та відтворення аудіо, підтримку більш ніж одного формату відео, сприйняття стрімів з Youtube. Хід роботи основне завдання + підтримка більш ніж одного формату відео + відео з фреймів-різниць Алгоритми апроксимації руху Farneback method opt.flow calculation Encoder In [1]: import numpy as np import cv2 import time # Get our sample video cap = cv2.VideoCapture('walking.avi') # Define the codec and create VideoWriter object fourcc = cv2.VideoWriter fourcc(*'XVID') frame width = int(cap.get(3)) frame height = int(cap.get(4)) out = cv2.VideoWriter('Farn_encode.avi', fourcc, 20.0, (frame_width, frame_height)) # we will aproximate only even frames, while odd frms won't be change odd frame = **True** ret, frame1 = cap.read() prvs = cv2.cvtColor(frame1,cv2.COLOR BGR2GRAY) hsv = np.zeros like(frame1) hsv[...,1] = 255t start = time.time() while (cap.isOpened()): ret, frame2 = cap.read() if not ret: break next = cv2.cvtColor(frame2,cv2.COLOR BGR2GRAY) flow = cv2.calcOpticalFlowFarneback(prvs,next, None, 0.5, 3, 15, 3, 5, 1.2, 0) # Create an image representing our opt.flow mag, ang = cv2.cartToPolar(flow[...,0], flow[...,1]) hsv[...,0] = ang*180/np.pi/2hsv[...,2] = cv2.normalize(mag, None, 0, 255, cv2.NORM MINMAX) frame_flow_rgb = cv2.cvtColor(hsv,cv2.COLOR_HSV2BGR) cv2.imshow('actual footage', frame2) cv2.imshow('pure optical flow', frame flow rgb) # write frame to output video. if(odd frame): out.write(frame2) odd frame = False else: out.write(frame flow rgb) odd frame = **True** if cv2.waitKey(1) & 0xFF == ord('q'):break # Aaaaand... go to the next frame! prvs = next t end = time.time() # time metric Farn_encode_time = t_end - t_start cap.release() out.release() cv2.destroyAllWindows() Decoder In [2]: import numpy as np import cv2 import time # Get our sample video cap = cv2.VideoCapture('walking.avi')

Define the codec and create VideoWriter object

prvs = cv2.cvtColor(frame1,cv2.COLOR BGR2GRAY)

next = cv2.cvtColor(frame2,cv2.COLOR BGR2GRAY)

flow[:,:,1] += np.arange(h)[:,np.newaxis]

cv2.imshow('warped', frame warped)

out.write(frame warped)

if cv2.waitKey(1) & 0xFF == ord('q'):

Aaaaand... go to the next frame!

out = cv2.VideoWriter('Farn decode.avi', fourcc, 20.0, (frame width, frame height))

flow = cv2.calcOpticalFlowFarneback(prvs,next, None, 0.5, 3, 30, 3, 5, 1.2, 0)

flow = -flow # here we invert the flow in order to get the previous image

frame warped = cv2.remap(frame2, flow, None, cv2.INTER LINEAR)

we will aproximate only even frames, while odd frms won't be change

fourcc = cv2.VideoWriter fourcc(*'XVID')

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

odd frame = True

hsv[...,1] = 255

t_start = time.time()
while(cap.isOpened()):

ret, frame1 = cap.read()

hsv = np.zeros_like(frame1)

if not ret: break

h, w = flow.shape[:2]

if(odd frame):

prvs = next

t_end = time.time()

cv2.destroyAllWindows()

time metric

cap.release()
out.release()

Encoder

import cv2
import time

import numpy as np

odd_frame = **True**

hsv[...,1] = 255

t_start = time.time()
while(cap.isOpened()):

ret, frame1 = cap.read()

hsv = np.zeros like(frame1)

if not ret: break

if(odd frame):

prvs = next

t end = time.time()

cv2.destroyAllWindows()

Get our sample video

odd frame = True

hsv[...,1] = 255

t_start = time.time()
while(cap.isOpened()):

if not ret: break

frame_width = int(cap.get(3))
frame height = int(cap.get(4))

ret, frame1 = cap.read()

hsv = np.zeros like(frame1)

ret, frame2 = cap.read()

h, w = flow.shape[:2]

if(odd frame):

prvs = next

t end = time.time()

cv2.destroyAllWindows()

def get diff(v1, v2):

time metric

cap.release()
out.release()

Точність

In [6]: import cv2

else:

flow[:,:,0] += np.arange(w)

out.write(frame2)
odd frame = False

odd frame = **True**

Dual decode time = t end - t start

Порівняння алгоритмів

Get our sample video

origin = cv2.VideoCapture(v1)
aprox = cv2.VideoCapture(v2)

frame_width = int(origin.get(3))
frame height = int(origin.get(4))

compute difference

color the mask red

out.write(difference)

origin.release()
aprox.release()
out.release()

In [7]: get_diff('walking', 'Farn_decode')

Швидкість роботи

In [10]: Farn_encode_time

Out[10]: 225.7192506790161

In [11]: Farn_decode_time

In [12]: Dual_encode_time

Out[12]: 2637.011543273926

Out[13]: 2381.591493368149

Аналіз

точок зображення.

форми

Dual_decode_time

Out[11]: 223.70712995529175

In [9]:

In [13]:

cv2.destroyAllWindows()

get_diff('walking', 'Dual_decode')

get_diff('Dual_decode', 'Farn_decode')

ret1, frame1 = origin.read()
ret2, frame2 = aprox.read()

Створимо відео на якому зображено різницю між двома файлами.

Define the codec and create VideoWriter object

while(origin.isOpened() and aprox.isOpened()):

if (ret1 == False or ret2 == False): break

difference = cv2.subtract(frame1, frame2)

difference[mask != 255] = [0, 0, 255]

Conv hsv Gray = cv2.cvtColor(difference, cv2.COLOR BGR2GRAY)

З отриманих візуальних різниць та метрик швидкодії можемо зробити такі висновки:

функціоналу, що є сумою повної варіації (Total Variance) векторного поля и та норми

використовується як початкові точки на наступних рівнях [4].

[5] https://vision.middlebury.edu/flow/eval/results/results-a1.php

[6] https://vision.middlebury.edu/flow/eval/results/results-e1.php

Використані джерела

перевищували відповідні метрики для методу Farneback.

роздільною здатністю і, відповідно, розміром кадрів.

out = $cv2.VideoWriter(f'diff_({v1})_&&_({v2}).avi', fourcc, 20.0, (frame_width, frame_height))$

ret, mask = cv2.threshold(Conv_hsv_Gray, 0, 255,cv2.THRESH_BINARY_INV | cv2.THRESH OTSU)

• Farneback показав трохи менші візуальні різниці, в порівнянні з методом Dual TV-L1, хоча різниця для тестових відео виявилась

Варто зазначити, що для значного поліпшення швидкодії цих методів можна подавати на вхід зображення (кадри) меншого розміру,

Обидва алгоритми, Dual TV-L1 та Farneback є алгоритмами для обчислення щільного (dense) optical flow. Методи апроксимації руху за допомогою Optical Flow демонструють значно кращу точність результатів апроксимації в порівнянні, наприклад, з block matching методами [1]. Вони також демонструють більшу стійкість при роботі із зображеннями зі спотвореннями, так, optical flow методи здатні

Dense optical flow алгоритми, зокрема, працюють із щільним оптичним потоком, тобто вираховують зсув не окремих, а усіх точок зображення, на відміну від sparce або вибіркового оптичного потоку. Вибірковий потік зазвичай вираховується швидше, за щільний, проте деякі методи маєть не таку й значну різницю в швидкодіх та деякі задачі потребують знаходження оптичного потоку для усіх

Обидва методи працюють з показником інтенсивності точки, тобто яскравості або кольору (для кольорових зображень). Метод Dual

 $E(\mathbf{u}) = \int_{\Omega} |\nabla u_1| + |\nabla u_2| + \lambda |\rho(\mathbf{u})|$

 $I(x) = x^T A x + b^T x + c$

Варто відзначити, що обидва методи добре спрацьовують при невеликих зсувах і мають проблеми при зсувах, більших за 1 піксель. В такому випадку рекомендується використовувати multi-scaling: будувати «піраміду» з зображень із меншою роздільною здатністю,

При роботі із зображеннями із спотвореннями метод Farneback демонструє гірші [4] показники середньої кутової похибки (Average

[1] Zhi Liu, Jianwen Luo "Performance comparison of optical flow and block matching methods in shearing and rotating models", Proc. SPIE

10139, Medical Imaging 2017: Ultrasonic Imaging and Tomography, 1013917 (13 March 2017); https://doi.org/10.1117/12.2253689

[3] G. Farneback, "Fast and accurate motion estimation using orientation tensors and parametric motion models," Proceedings 15th

[4] J. S. Perez, E. Meinhardt-Llopis, and G. Facciolo. "TV-L1optical flow estimation", IPOL Journal, 3:137–150, 2013

[2] Josh Harguess, Chris Barngrover, and Amin Rahimi "An analysis of optical flow on real and simulated data with degradations", Proc. SPIE 10199, Geospatial Informatics, Fusion, and Motion Video Analytics VII, 1019905 (1 May 2017); https://doi.org/10.1117/12.2265850

International Conference on Pattern Recognition. ICPR-2000, Barcelona, Spain, 2000, pp. 135-139 vol.1, doi: 10.1109/ICPR.2000.905291.

ніж оригінал. Алгоритм спочатку проходить по найнижчому рівню піраміди (із найменшою роздільною здатністю) і результат

Angle Error) та середньої похибки кінцевої точки (Average End-Point Error) на датасетах Middlebury [5][6] та Sintel.

в той час як метод Farneback [3] ґрунтується на ідеї апроксимації зміни інтенсивності в околі точки за допомогою квадратичної

TV-L1 ґрунтується на модифікації функціоналу, описаного в методі Horn-Schunck для врахування можливих розривів або переривчастості в оптичному потоці шляхом зміни квадратичних коефіцієнтів [4], і може бути представлений як мінімізація

• Dual TV-L1 був значно повільнішим в роботі, для деяких тестових відео метрики швидкодії цього методу в кілька разів

наприклад зменшувати їх вдвічі по обох осях. В нашому прикладі це було наочно видно при подачі на вхід відео із меншою

витримувати поворот об'єкта до 4° та деформацію зсуву до 6 відсотків без втрат в точності апроксимації.

fourcc = cv2.VideoWriter_fourcc(*'XVID')

break

time metric

cap.release()
out.release()

Decoder

In [4]: import numpy as np
import cv2
import time

ret, frame2 = cap.read()

hsv[...,0] = ang*180/np.pi/2

write frame to output video.

out.write(frame flow rgb)

if cv2.waitKey(1) & 0xFF == ord('q'):

Aaaaand... go to the next frame!

out.write(frame2)
odd frame = False

odd_frame = **True**

Dual encode time = t end - t start

cap = cv2.VideoCapture('walking.avi')

fourcc = cv2.VideoWriter fourcc(*'XVID')

Define the codec and create VideoWriter object

prvs = cv2.cvtColor(frame1,cv2.COLOR BGR2GRAY)

next = cv2.cvtColor(frame2,cv2.COLOR BGR2GRAY)

flow = optical flow.calc(prvs, next, None)

flow[:,:,1] += np.arange(h)[:,np.newaxis]

cv2.imshow('warped', frame warped)

out.write(frame_warped)

if cv2.waitKey(1) & 0xFF == ord('q'):

Aaaaand... go to the next frame!

optical flow = cv2.optflow.DualTVL1OpticalFlow create()

frame warped = cv2.remap(frame2, flow, None, cv2.INTER LINEAR)

out = cv2.VideoWriter('Dual decode.avi', fourcc, 20.0, (frame width, frame height))

flow = -flow # here we invert the flow in order to get the previous image

cv2.imshow('actual footage', frame2)

Get our sample video

frame_width = int(cap.get(3))
frame height = int(cap.get(4))

In [3]:

flow[:,:,0] += np.arange(w)

out.write(frame2)
odd frame = False

odd frame = **True**

Farn_decode_time = t_end - t_start

Dual TV L1 method opt.flow calculation

cap = cv2.VideoCapture('walking.avi')

fourcc = cv2.VideoWriter fourcc(*'XVID')

Define the codec and create VideoWriter object

prvs = cv2.cvtColor(frame1,cv2.COLOR BGR2GRAY)

next = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)

flow = optical flow.calc(prvs, next, None)

Create an image representing our opt.flow

cv2.imshow('pure optical flow', frame_flow_rgb)

optical_flow = cv2.optflow.DualTVL1OpticalFlow_create()

mag, ang = cv2.cartToPolar(flow[...,0], flow[...,1])

frame_flow_rgb = cv2.cvtColor(hsv,cv2.COLOR_HSV2BGR)

hsv[...,2] = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)

out = cv2.VideoWriter('Dual_encode.avi', fourcc, 20.0, (frame_width, frame_height))

we will aproximate only even frames, while odd frms won't be change

ret, frame2 = cap.read()

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

«Інститут прикладного системного аналізу»

Кафедра математичних методів системного аналізу

Звіт

про виконання лабораторної роботи №2

з дисципліни

«Розпізнавання образів»

Київ - 2020

Виконали: студенти IV курсу

Перевірила:

групи КА-76

Панасюк Я.І.

Іванов С.І.

та групи КА-74

Дідковська М.В.